INTERNATIONAL JOURNAL OF ROBUST AND NONLINEAR CONTROL
Int. J. Robust Nonlinear Control 2002; **12**:1009–1028 (DOI: 10.1002/rnc.729)

Gradient-based parameter optimization for systems containing discrete-valued functions

Edward Wilson[1,*,†,‡] and Stephen M. Rock[2,§]

[1] *Intellization, 454 Barkentine Lane, Redwood Shores, CA 94065, U.S.A*
[2] *Stanford University, Aerospace Robotics Laboratory, Stanford, CA, 94305, U.S.A*

SUMMARY

Gradient-based parameter optimization is commonly used for training neural networks and optimizing the performance of other complex systems that only contain continuously differentiable functions. However, there is a large class of important parameter optimization problems involving systems containing discrete-valued functions that do not permit the direct use of gradient-based methods. Examples include optimization of control systems containing discrete-level actuators such as on/off devices, systems with discrete-valued inputs and outputs, discrete-decision-making systems (accept/reject), and neural networks built with signums (also known as hard-limiters or Heaviside step functions) rather than sigmoids. Even if most of the system is continuously differentiable, the presence of one or more discrete-valued functions will not allow gradient-based optimization to be used directly. A new algorithm, 'noisy backpropagation,' is developed here, as an extension of backpropagation, which solves this problem and extends gradient-based parameter optimization to permit application to systems containing discrete-valued functions. Moreover, the modification to backpropagation is small, requiring only (1) replacement of the discrete-valued functions with continuously differentiable approximations, and (2) injection of noise into the smooth approximating function on the forward sweep during training. Noise injection is the key to reducing the round-off error created when the discrete-valued functions are replaced after training. This generic approach is applicable whenever gradient-based parameter optimization is used with systems containing discrete-valued functions; it is not limited to training neural networks. The examples in this paper demonstrate the use of noisy backpropagation in training two different multi-layer signum networks and in training a neural network for a control problem involving on-off actuators. This final example includes implementation on a laboratory model of a 'free-flying space robot' to validate the realizability and practical utility of the method. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: optimization; neural network; backpropagation; hard-limiter

*Correspondence to: Edward Wilson, Intellization, 454 Barkentine Lane, Redwood Shores, California 94065, U.S.A.
†E-mail: ed.wilson@intellization.com
‡This work was performed as part of E. Wilson's doctoral research at the Stanford Aerospace Robotics Laboratory, Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305, U.S.A.
§S. Rock is with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA 94305.

Contract/grant sponsor: NASA Coop-Agreement; contract/grant number: NCC 2-333-S18.
Contract/grant sponsor: Department of Airforce; contract/grant number: F49620-92-J-0329.
Contract/grant sponsor: Hughes Aircraft Company

 Accepted 17 May 2002

1. INTRODUCTION

1.1. Problem statement

Optimization methods that use gradient information often converge much faster than those that do not, and using the backpropagation (BP) algorithm [1,2] to get this gradient information for training neural networks (NNs) has made them useful in many applications. However, BP's requirement of continuous differentiability, not only for the network itself, but for anything through which the error is backpropagated (e.g. the plant model in a control problem), limits its applicability. In particular, parameter optimization of systems containing discrete-valued functions (DVFs) represent a large class of important problems that cannot be optimized using existing gradient-based parameter optimization (GBPO) methods. When optimizing these systems, alternative methods which are not restricted to continuously differentiable functions, such as unsupervised learning, simulated annealing, integer programming, downhill simplex, or genetic algorithms may be used. However, these alternatives are usually significantly slower because they do not use gradient information.

GBPO refers to the optimization of parameters which are updated iteratively based on estimates of the cost-parameter gradient (i.e. the partial derivatives of the optimization cost function with respect to each parameter). In NN training, the parameters are the NN weights and biases, and BP is the algorithm used to obtain the estimated cost-parameter gradient, $\partial \text{cost}/\partial W$, where W is a vector containing all weights and biases to be optimized. There are numerous algorithms to calculate the parameter update based upon the estimated cost-parameter gradient.

BP uses the chain rule to calculate partial derivatives throughout a system, based on costs evaluated (usually) at the system output. The partial derivatives of the cost function with respect to variables and parameters are calculated in a 'backward propagation' from the cost measurement point (e.g. terminal condition in a control optimization problem) through each of the intermediate functions (e.g. plant model, control gains, or control inputs) that affect the cost.

The term 'backpropagation' is commonly used to refer to NN training, but in this paper it also refers to use of the chain rule for efficient calculation of derivatives in any GBPO problem. Also, 'optimization' in this paper refers to the optimization of parameters in a control or decision-making system as well as to the training of NNs (optimizing the weights and biases that define the NN's functionality). While noisy backpropagation is equally applicable to GBPO in NN and non-NN applications, the examples developed in this paper all involve NNs.

It is common for a problem to be well suited for GBPO, except for the presence of a few DVFs. The on-off-thruster control application presented in Section 4 is a prime example: a NN (continuously differentiable) produces an output that is discretized (with a non-differentiable function) and then passed through a model of the robot–thruster system (continuously differentiable) before the performance can be evaluated and used for training. Except for the on-off thrusters, this problem is continuously differentiable, and therefore would be well suited for GBPO. This situation is common when DVFs are involved: the DVFs represent a small portion of the overall system, but the problem they present for GBPO is formidable. Specifically, conventional BP will not work when one or more DVFs exist on the functional path between the parameters to be optimized and the cost function.

1.2. Related research

The problem addressed in this paper is related to a similar, but more specific problem that has received attention in the field of NNs dating back to 1962 [3]: training multi-layered networks of hard-limiting neurons ('signum networks'). In particular, several techniques have been developed, but each involves reducing the problem to a special case. Examples include:

- Learning algorithms for single-layer networks date back to 1960, with Widrow's ADALINE (ADAptive LInear NEuron) [4] and Rosenblatt's Perceptron [5]. Neither of these methods extends directly to multiple layers.
- MADALINE (Many ADAptive LInear NEurons) Rule I was a two-layer network (one hidden layer) that had a trainable first layer, but the second layer was a fixed logic operation, such as OR, AND, or MAJ (majority) [3]. In MADALINE Rule II, Winter [6] used a heuristic approach that had limited success at training a two-layer network of hard-limiters (ADALINEs). These methods may be classified as 'error-correction rules' rather than 'steepest-descent rules' (gradient-based) [4].
- In research aimed directly at using gradient-based learning for multi-layer signum networks, Bartlett and Downs [7] use weights that are random variables, and develop a training algorithm based on the fact that the resulting probability distribution is continuously differentiable. The algorithm is limited to one hidden layer, requires all inputs to be or 1 or -1, and needs extra computation to estimate the gradient.
- Backpropagation [1,2] solved the multi-layer neural network training problem by replacing the previously proposed hard-limiting (signum) neurons with continuously differentiable (sigmoid) neurons. However, the performance may degrade significantly if the sigmoids are replaced after training with signums.

With noisy backpropagation, GBPO has been extended to apply to the general problem of optimizing systems with DVFs, of which training signum NNs is a special case. Just as BP can be applied to arbitrary functional architectures as long as information flow is in one direction and all functions are continuously differentiable, noisy backpropagation is not limited to single hidden layer, etc.

1.3. Round-off error

A commonly used method to optimize systems containing DVFs is to approximate the DVFs with linear functions or smooth sigmoids during the learning phase, and switch to the true discontinuous functions at run-time; however, this is inaccurate due to the round-off errors introduced by the DVFs. This problem is most severe when multiple DVFs are present, as illustrated by the simple example in Section 2.2, and the more complex examples in Sections 3 and 4. The aforementioned method is exactly equivalent to Noisy Backpropagation without the addition of noise, (or as will be discussed, $N = 0$) allowing for a direct evaluation of the benefits of noise injection.

Noisy Backpropagation solves this problem by using artificially injected noise to produce approximations of DVFs that can be used in GBPO yet do not introduce significant round-off error at run-time.

This use of noise is completely new. It is however somewhat analogous to a number of applications where noise has been studied and used to improve performance. Noise as it relates to digital sampling [8], dither in control applications, the human vision system, prevention of

overfitting in NN training [9], simulated annealing, and increasing the convergence rate of NN training [10] are discussed in Reference [11].

In contrast to previous applications of noise in NN training [9,10], here noise is injected into the signal immediately before the sigmoid calculation (as opposed to the inputs or the weights). Injecting noise at this point modifies the effect of the sigmoid during training, enabling gradient based optimization of DVFs, rather than to improving generalization or increasing the convergence rate.

1.4. Outline of paper

There are three major sections in this paper. In Section 2, the new technique of learning using 'noisy sigmoids' is explained and a training algorithm is derived. In Section 3, the utility of this method for training multi-layered networks of hard-limiters is demonstrated. Finally, in Section 4, its usefulness for complex control optimization problems is demonstrated by applying it to a thruster-control problem involving eight on-off thrusters controlling a 3-degree-of-freedom free-flying space robot.

2. A NEW TRAINING ALGORITHM: NOISY BACKPROPAGATION

Noisy Backpropagation extends BP to enable its use in the optimization of systems containing DVFs. The extension proposed is to approximate the DVFs with continuously differentiable functions and then to add noise during training at the input of each DVF-approximating function. No modification to the backward sweep or the weight update rule (that computes weight updates as a function of the estimated weight gradient, $\partial\,\mathrm{cost}/\partial W$) is required [11–13], meaning noisy backpropagation is compatible with many other modifications and improvements to BP.

The training algorithm, shown in Figure 1 for a single signum neuron, is summarized as follows:

- Replace the DVFs with continuously differentiable approximating functions during training. For example, replace signums with sigmoids.
- Inject noise immediately before the approximating functions (sigmoids) on the forward sweep (selecting the type and magnitude of the noise is discussed in Section 2.3).
- Use the exact same backward sweep as with standard BP to get the estimated cost-parameter gradient (e.g. for NN training, $\partial\,\mathrm{cost}/\partial W$).
- Apply the same parameter update rule (that computes parameter updates as a function of $\partial\,\mathrm{cost}/\partial\,\mathrm{parameter}$) as with standard BP (or a modification of BP).

Intuitively, the algorithm works because it meets two criteria:

Continuously differentiable: The function used to approximate the DVF is continuously differentiable, permitting backpropagation of the gradient estimate.

Accurate representation of DVF: The key problem of round-off error is addressed by the noise, which produces random outputs for DVF inputs in the sigmoid's transition region. During training, this randomness generally results in a higher error than would result if the sigmoid's saturation regions were used (in most cases, a random output will be worse than a well chosen one). The result is that *during optimization* the solution is penalized for using transition regions,

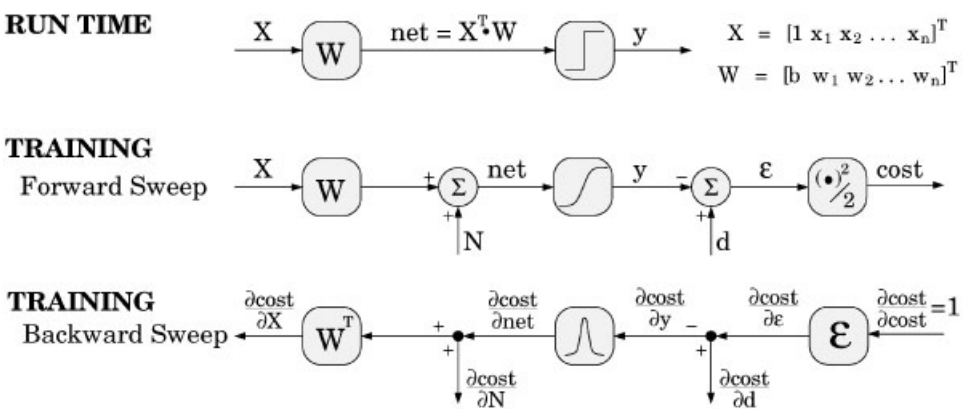

Figure 1. Noisy Backpropagation algorithm. The algorithm is shown here applied to the DVF in a single signum neuron. The procedure for gradient-based parameter optimization with DVFs in signum networks, sigmoid networks, and non-NN applications is the same.

so it adapts to avoid these regions. Then, when the original DVFs are replaced, the change in performance is minimal.

The algorithm and issues associated with its use are developed in the following sections.

2.1. *Training algorithm*

To help understand the algorithm, consider the block diagram shown in Figure 1. The first block diagram in Figure 1 shows a single hard-limiting neuron as it appears at run-time: a dot product with hard limiter. For simplicity in bookkeeping, the input, X, and weight, W, vectors are augmented to include the threshold bias for the output function. The next two diagrams show the neuron during training, where the signum has been replaced by a smooth sigmoid function. The input, X, is propagated through the forward sweep and the result compared with the desired output, d, finally resulting in an error, ε, and a cost. The derivative of this cost is calculated and propagated though the backward sweep, resulting in a $\partial\,\mathrm{cost}/\partial X$ to be propagated to more units upstream, and a $\partial\,\mathrm{cost}/\partial\,\mathrm{net}$ to be used in calculating $\partial\,\mathrm{cost}/\partial W$, which is used in the weight-update algorithm. The derivative of the sigmoid itself is calculated based on the sigmoid-input value before noise is added.

This is almost the same as training a standard neuron with BP; the only difference involves the injection of zero-mean noise, N, immediately before the sigmoid. A more complete derivation of the conventional BP algorithm can be found in several sources [1,2,4]. While the mechanics of the backward sweep is identical, different weight updates result because the forward sweep resulted in a different error.

Note that the noise injection does not corrupt the calculation of $\partial\,\mathrm{cost}/\partial W$, just as the desired signal does not. Using an unmodified backward sweep is not only the simplest thing to do, it does precisely the right calculations for estimating the weight gradient.

Just as with BP, Noisy Backpropagation extends to multiple layers in a NN or to multistage systems in optimal control [14].

2.2. Intuitive explanation, example

Before explaining why noisy backpropagation works, it is important to understand the problem it was designed to solve and specifically, the round-off error it was designed to reduce.

GBPO requires systems to be continuously differentiable, which in turn requires any function that is used to approximate a DVF to cover smoothly the region between the discrete values. During optimization, the system (NN, for example) may use its degrees of freedom to use the full range of this 'transition region,' possibly resulting in excellent performance with the smooth approximating functions in place.

However, when the DVFs are replaced during testing or at run-time, significant round-off error can arise. Round-off error can be most significant when multiple DVF outputs are combined; the total error will be worse if the individual DVF round-off errors are systematically added. Therefore, the purpose of the algorithm is to modify the optimization so that round-off error is reduced; this is accomplished by penalizing the use of the DVF-approximating-function's transition region during optimization.

A simple example illustrates the effect of round-off error. A signum network shown in Figure 2 is to be trained (i.e. the six weights and four biases shown are to be optimized) to approximate the sigmoidal function also shown in Figure 2. When the signums are replaced with sigmoids and the system is trained, mapping performance is near perfect. The three neurons all produce roughly the same output: a sigmoid rising from 0 to 1, with a slope of 0.25 at the inflection point, and with the transition region centered at zero. The outputs of these individual sigmoid neurons are almost perfectly superimposed in Figure 3 and when combined, result in a near perfect match with the desired total output. However, when the sigmoids are replaced with signums, the net result is roughly a Heaviside step function going abruptly from 0 to 3 at 0. This is an extreme case of round-off error, where the errors from each DVF add. The mapping error is shown as the shaded regions in Figure 3.

It would appear that by forcing the individual sigmoids to be sharper, making the DVF-approximating functions more closely resemble the DVFs, they would distribute themselves more evenly, reducing round-off error. It will be demonstrated that this is a byproduct of what happens when noisy backpropagation is applied. However, *manually* adjusting the sharpness of the DVF-approximating functions has no effect at all on the results, as can be seen clearly by

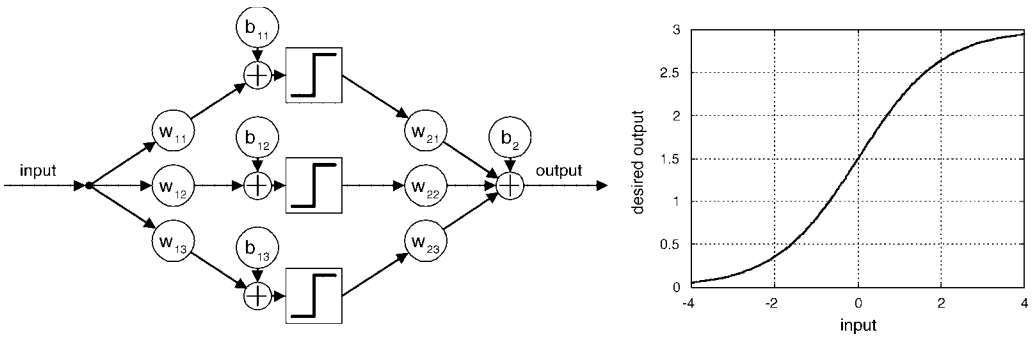

Figure 2. Example problem definition. The goal is to select the weights and biases $(w_{11}, w_{12}, \ldots, b_2)$ in the signum network shown so that the input–output mapping will match the desired mapping as closely as possible. The desired mapping is plotted to the right.

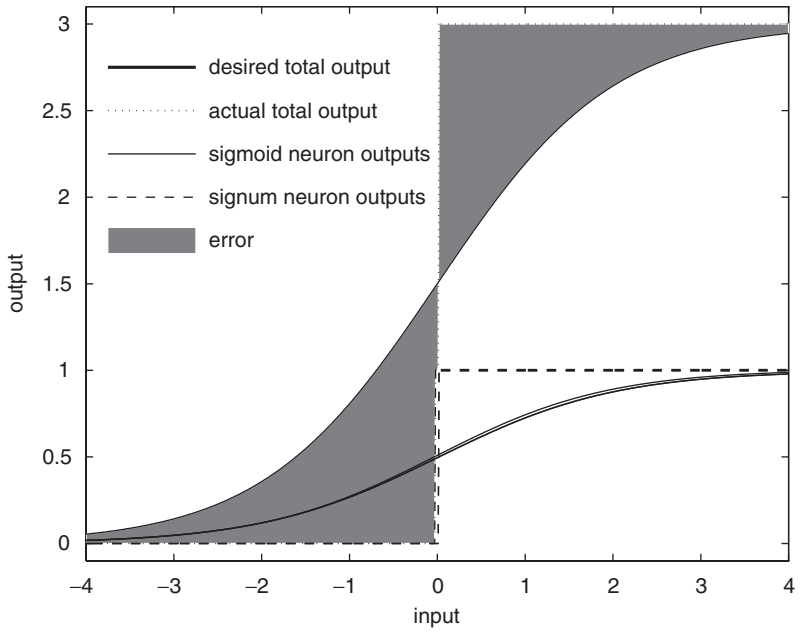

Figure 3. Training results with no noise added. With no noise added during training, significant roundoff error results, indicated by the shaded regions.

examining the sigmoid equation. The equation for a sigmoid that goes from 0 to 1 may be written as follows:

$$y = \frac{1}{1 + e^{-\beta WX}}$$

where y is the sigmoid output, W is the input weight vector, including bias, X is the input vector, and beta is a parameter that can be adjusted to change the sharpness of the sigmoid.

Considering WX the input and y the output, the sharpness (slope) is proportional to β. Since β is multiplied directly by W, during optimization, W will adapt to cancel *exactly* any sharpening effect the user was trying to create by selecting a high value for β. β is not an independent parameter, so in this research it has been fixed at a value of 1, effectively removing it.

In noisy backpropagation, the effect of noise injection is to randomize the output of the sigmoid in the transition region, as shown in Figure 4. In most cases, a randomized output will decrease the performance of the system. If the system (e.g. NN) possesses sufficient degrees of freedom (e.g. weights and biases), it may be able to achieve the desired input–output mapping while reducing the error due to randomization. Reducing the error due to randomization is achieved by avoiding use of the sigmoid transition regions where possible, since this is where the output is most affected by noise in the input.

The result is that noise injection effectively penalizes the network for choosing weights that use noisy neurons in their transition regions, leading to a sharpening of the neurons.

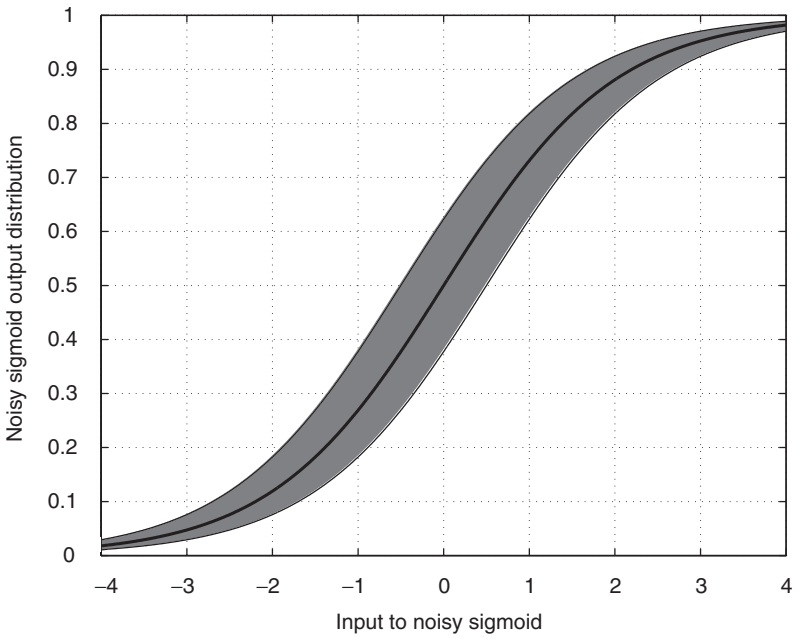

Figure 4. Input–output distribution for noisy sigmoid. Zero-mean, white, uniformly distributed noise with a maximum magnitude of 0.5 added to a sigmoid function randomizes the output in the transition region.

Introducing this penalty through noise injection is a simple procedure which does not require modification to the backward sweep in BP.

Since the transition region is avoided and saturation regions are favored where possible, when training stops and the DVFs (signums, for example) are replaced, the effect on system output is reduced. Noise injection caused the system parameters (NN weights and biases) to be adapted to minimize this round-off error.

The results of noise injection can be seen in Figure 5. This is the same example shown in Figure 2, except that uniformly distributed white noise with a magnitude of 0.5 (this is represented by the following expression: $N = 0.5$) was added to each sigmoid during training. $N = 0.5$ corresponds to the noisy sigmoid shown in Figure 4.

With noise added, the gradient-based optimization is guided towards a solution that achieves the desired input–output mapping while minimizing the use of each sigmoid's transition region. In doing this, the individual sigmoid outputs have sharpened and rearranged themselves as shown in Figure 5. When tested with sigmoids in place, performance is slightly worse than for the network trained with $N = 0$, but still excellent. The importance of the sharpening and rearranging of the sigmoids is seen when they are replaced with signums: now the discrete steps do not all occur at $input = 0$. The mapping performance is significantly better, evidenced by the reduced shaded area in Figure 5.

Figure 6 shows the input–output distribution (during training) before and after optimization using noisy backpropagation. The smaller variance of the plot on the right indicates that the algorithm has succeeded in reducing the variability in the mapping. Examining the individual neuron outputs in Figures 3 and 5 shows that this was accomplished by sharpening each of the

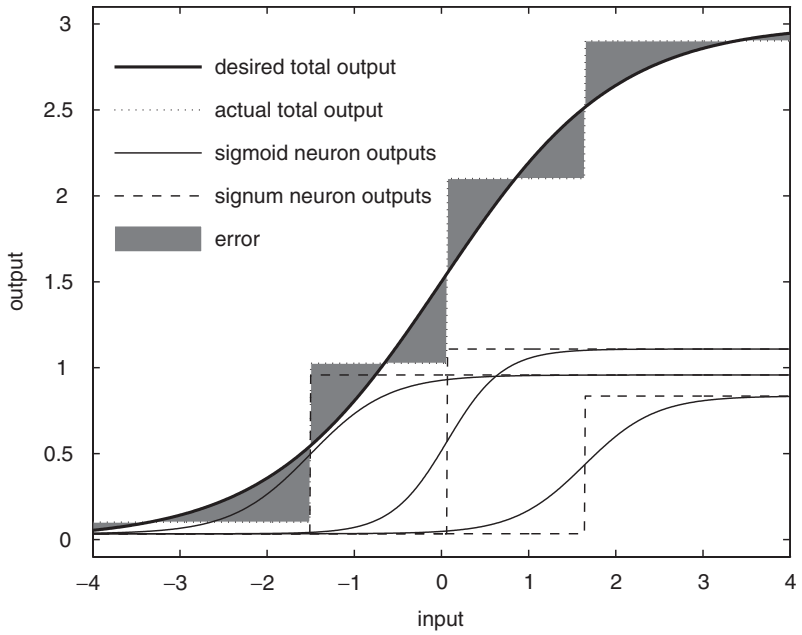

Figure 5. Training results with noise added. With noise added during training, the individual sigmoids are sharpened and have distributed themselves away from the origin. When sigmoids are replaced with signums, roundoff error is reduced.

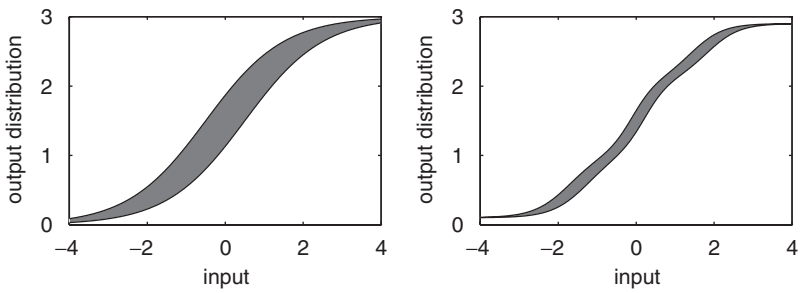

Figure 6. Noisy backpropagation effectively penalizes the use of transition regions. The plot on the left shows the input–output distribution during training with noisy sigmoids for the network from Figure 2 after the 100th epoch, immediately after noise is introduced. The plot on the right shows the distribution after the 10 000th epoch. Optimization with noisy backpropagation has reduced the error by sharpening the individual sigmoids, thus avoiding the transition region where possible.

sigmoids, thereby reducing the range of the transition region. To achieve the desired mapping with sharper sigmoids, the individual neuron's centers were moved.

The motion of the individual sigmoid transition regions as training progresses is shown in Figure 7. The horizontal axis shows the training iteration ('epoch'). The vertical axis shows the location of each sigmoid's centre—the inflection point of its input–output function. Noise was

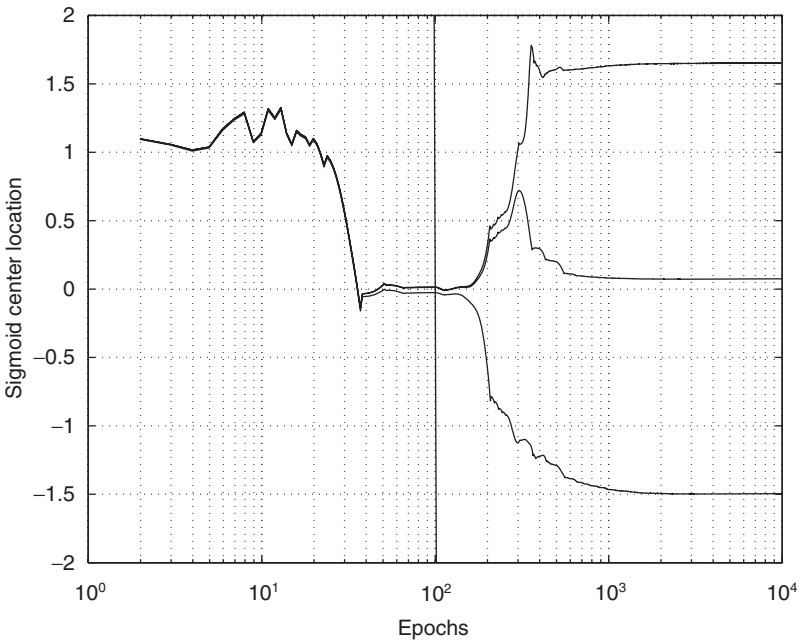

Figure 7. Position of sigmoid centres as training progressed. With no noise added between 1 and 100 epochs, sigmoid centres all moved to near zero. With noise level increased to $N = 0.5$ between 100 and 10 000 epochs, sigmoid centers moved to create the stepped mapping shown in Figure 5.

set to zero for the first 100 iterations, and the system quickly converged to the result shown in Figure 3, with sigmoid centres all near zero. After the 100th iteration, $N = 0.5$, leading to the results shown in Figure 5. If training continued, the solution would eventually converge on a perfectly symmetric distribution of sigmoid centres. The complete MATLAB [15] source code used in this example is available via Internet [16].

This simple example highlights the problem of round-off error that occurs when DVF-approximating functions are replaced by the actual DVFs. It also demonstrates the effectiveness of noisy backpropagation in reducing it.

2.3. Selection of noise level

In the previous example, uniformly distributed zero-mean white noise with a maximum magnitude of 0.5. ($N = 0.5$) was shown to produce near optimal results. Fortunately, in most cases, the exact noise level and type selected is not critical; however, some important considerations will be mentioned here.

Since the purpose of the noise is to disturb the system and make the transition regions unattractive for optimization, the specific noise distribution used is not critical. Successful experiments were run with uniform, Gaussian, and binary (either +max or −max) noise distributions. Uniform distributions were selected only because they facilitated plotting and visualization of how the algorithm works. Zero-mean noise was used to prevent any unnecessary

bias. White noise was used to prevent any unwanted effects on the convergence of the optimization algorithm. The only remaining parameter is the magnitude of the noise.

Since noise adds a random element to the system, using the lowest noise level required to achieve the desired level of performance will usually result in the fastest convergence. However, as will be shown in an example in Section 3 for systems with bi-level DVFs, the noise level can be increased an order of magnitude beyond what is required with minimal degradation.

When multi-level DVFs are used, as seen in Figure 8, there *is* an upper limit to the noise level: too much noise may cause the individual sigmoids to overlap, which in this example would blur out the middle level. The specific level of noise at which this effect begins depends upon the sharpness of the sigmoids and the discrete values approximated. In Figure 8, with a sharpness factor of 4 (slope at midpoint = 4) and one unit between discrete levels $(-1, 0, 1)$, this effect begins around $N = 0.2$ and is significant at around $N = 0.3$. In the figure, $N = 0.15$. The effects of this on training will be shown in Section 4 and Figure 16.

The sufficient noise level for bi-level DVFs or the optimal noise level for multi-level DVFs can usually be chosen by examining a plot of the DVF-approximating function and estimating how much noise is needed to cover the transition region without being excessive. That is, the noise level, N, should be chosen on the same order as the width of the transition region, without causing overlap among noisy sigmoids. In the tri-level DVF discussed above, the optimal noise level could have been predicted by sketching the limits of the noise-altered function (the shaded region in Figure 16) and determining at what point the middle region (input $= 0 \rightarrow$ output $= 0$)

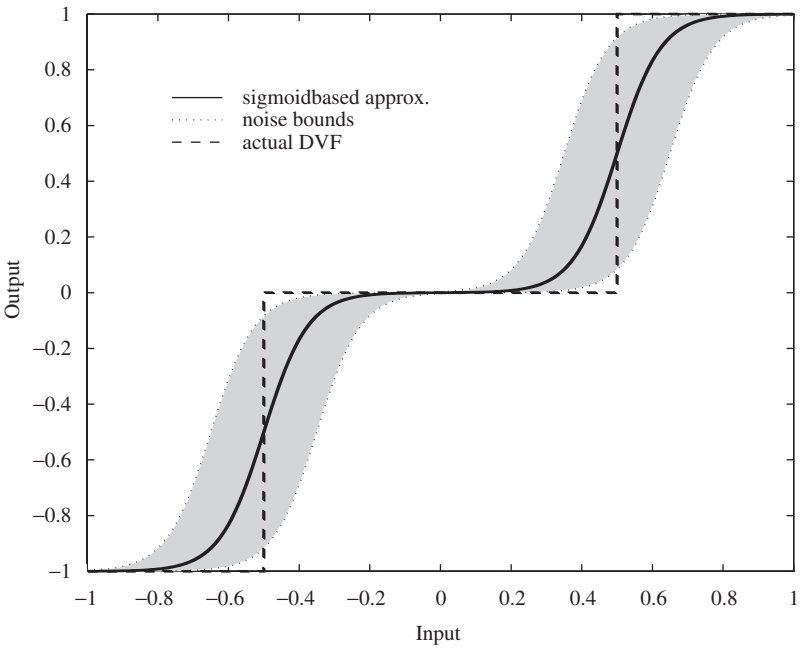

Figure 8. Approximation of tri-level DVF using noisy sigmoids. When DVFs with more than two levels are approximated, the noise level must be chosen to cover most of the transition regions without covering the saturation regions.

becomes affected by the noise. Use of this graphical method for the initial noise level estimate, followed by experimental iteration if necessary, has proven successful in several applications.

Figure 9 shows how the noisy sigmoid output distribution changes as the noise level increases: with no noise, only a single output can result; but as noise increases to cover most of the transition region, the output distribution approaches that of a hard-limiting function. Excessively high noise levels such as these are not necessary, but the figure shows graphically how the output of a noisy sigmoid approaches that of a signum as noise increases.

2.4. Application considerations, extensions

Sigmoid functions are used to build the DVF-approximating functions used in this research. However, any continuously differentiable function may be used, including any of the various types of sigmoids. If bi-level DVFs are used, the sharpness of the approximating function does not matter, as explained above.

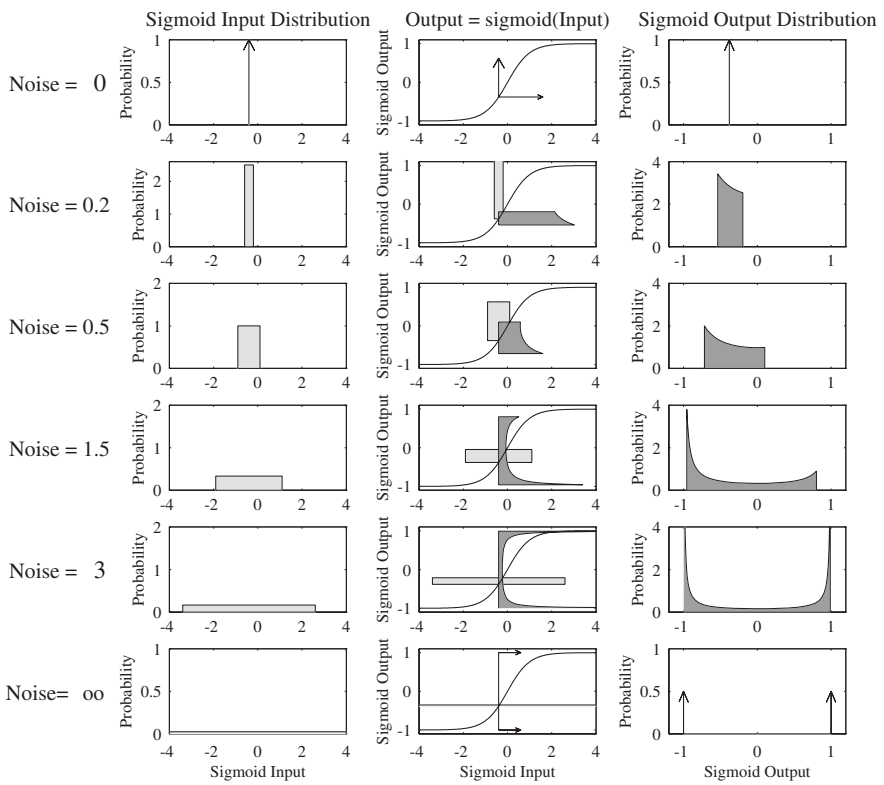

Figure 9. Effect of noise level on sigmoid output distribution. Lightly shaded region in column 1 represents the sigmoid input probability distribution (in this case, -0.3 + uniformly distributed noise). Darkly shaded region in column 3 is the sigmoid output distribution (from -1 to 1). Each distribution has an area of 1. Input and Output are plotted together in column 2 to show how the sigmoid produces this input–output relationship. As noise level increases, and the input distribution spreads out, the sigmoid output approaches that of a hard-limiter, while remaining differentiable.

A DVF whose output can take on n discrete values may be approximated by a linear combination of $n-1$ sigmoid functions. A tri-level DVF example is presented in Section 4. In this case, the sharpness of the approximating function does need to be selected carefully. The consideration here is that the sigmoids should be sharp enough to keep the saturated regions distinct from the transition regions; at the same time, if sharpness is too high, the derivative of the function will be very small in the saturated regions, possibly resulting in a slower convergence rate. Alternatively, the sigmoid-based DVF-approximating function may be developed through a supervised training technique using standard BP. In this case, adding a complexity cost [17,18] will keep the weights small, and will systematically limit the sharpness.

One concern is the attenuating effect of the derivative-of-sigmoid function. When back-propagated through many layers of near-saturated sigmoids, the error signal is attenuated and can lead to slow learning. Although important in conventional BP, this effect is an even greater issue with noisy backpropagation, because the sigmoid functions are used in their saturated regions, where the derivative is smaller. To handle this problem, it may be necessary to increase gradually the noise level; slowly pushing the outputs from the linear region to the hard-limits, rather than all at once. However, since all the experiments presented here had a single layer of discontinuity, no such gradual increase was required.

The randomness introduced with the addition of noise can make learning slow because of the reduction in signal-to-noise ratio in the weight gradient estimation. Batch-learning, using the exact same training set (considering the 'training set' to include the 'input set' and 'noise set') from one epoch to the next was used to generate all results presented here. Freezing the training set and noise set defines a fixed, deterministic cost hyper-surface. With a fixed cost function, on-line tuning of momentum and learning rate can be applied to improve dramatically the convergence rate. The MATLAB source code referred to earlier contains these improvements [16]. Many of the standard modifications to BP learning may be applied (for example, second order methods, etc.).

Another area where this method has potential is for optimization problems that have discrete valued parameters. For example, a design optimization problem where the task is to select the right number of bolts, pipe diameter, or gear ratio from a finite set of discrete-valued options. It is expected that noisy backpropagation will extend well to this family of problems (personal communication with Timothy W. McLain, 1993).

Also, many data-mining applications involve some sort of a discrete decision event. For example, a credit scoring system may decide among three options: accept, reject, or pass on to a human evaluator. A commonly used method for such problems is to develop a classifying system to assign each application to one of the three categories. However, a more advanced system using noisy backpropagation to represent the discrete-valued decision making step could allow optimization of the entire credit scoring process, including effects resulting from the decision.

3. APPLICATION TO TRAINING MULTI-LAYER SIGNUM NETWORKS

In the next two sections, the implementation of noisy backpropagation will be demonstrated by applying it to two very different problems: training a NN built with hard-limiting neurons, and optimizing the nonlinear thruster-mapping component of a control system for on-off thrusters

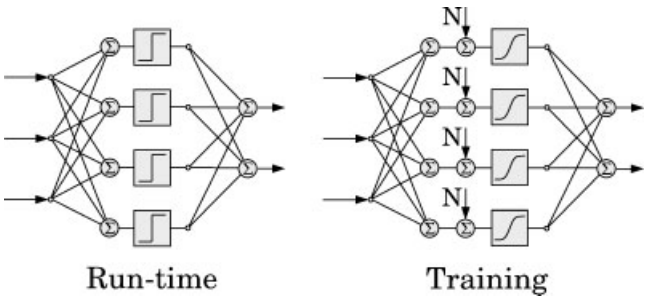

Figure 10. A multi-layer signum network, seen at run-time and during training.

on a space robot. Noisy backpropagation will be shown to apply to both of these problems without requiring any special modification.

In this section, noisy backpropagation is used to train a NN built with hard-limiting neurons. Figure 10 summarizes the method: during training, replace each hard-limiter with a sigmoid and zero-mean independent noise source.

An adaptive 3-5-4 signum network is trained to emulate the input–output mapping defined by a second, randomly generated, 3-10-4 sigmoid network. Fewer hidden neurons are used in the adaptive network to ensure that overfitting will not introduce unnecessary complications. The 3-10-4 network's fixed weights were randomly chosen between -2 and 2.

Performance is shown in Figure 11. Each dot on the graph represents the final performance after a full training run (10 000 epochs or until a local minimum was reached). Seven values for noise level were chosen, and since a global optimum is not guaranteed, ten different sets of initial weights were used at each noise value. The shaded region on the graph indicates the range of performance from all of the runs.

With noise level, $N = 0$, performance is good for the sigmoid network, but when the signums are reintroduced at run-time, the error increases dramatically.* One point is off the graph at a mapping error of over 6 units. As noise increases, mapping error on the sigmoid network increases, as expected, but error on the signum-network-performance decreases, and approaches the sigmoid-network-performance. Weight magnitude and neuron activation distribution plots confirm that as noise increases, the noisy sigmoids behave like hard-limiters (greater values in the first weight matrix, $W1$, effectively sharpen the sigmoids). These activation distributions could not have been achieved by manually increasing the sharpness of the sigmoids: this would have had *zero* net effect, since the network would adapt the first layer weights to counteract *exactly* the sharpness increase.

4. APPLICATION TO ROBOT THRUSTER CONTROL

In order to demonstrate the applicability of this new training procedure to a real-world complex control problem, it was applied to a thruster mapping problem characteristic of spacecraft [19].

*$N = 0$ represents the common method of replacing DVFs with smooth functions during training—it is used as a baseline for evaluating the performance improvements from the additional step of noise injection.

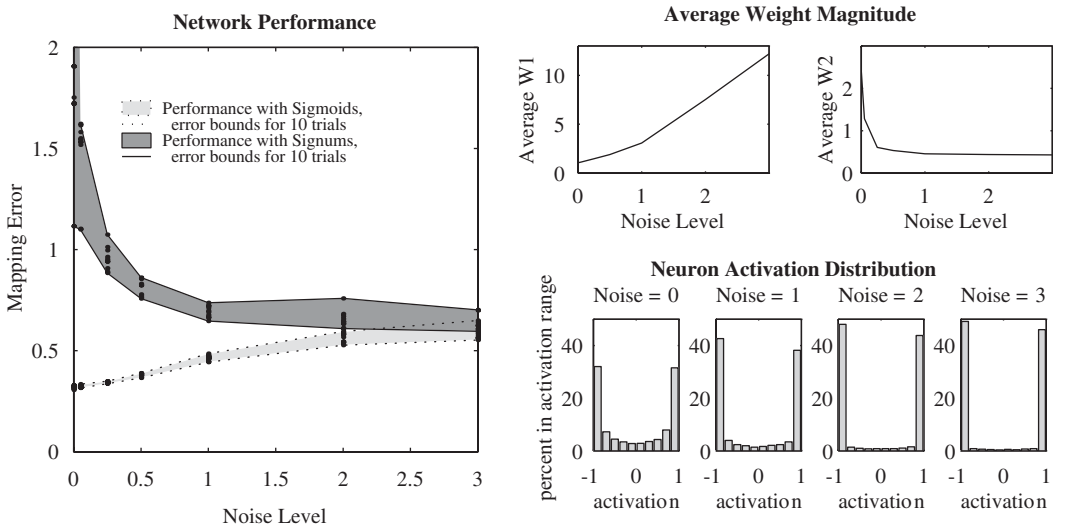

Figure 11. Training a Signum Network using Noisy Backpropagation. Left: with higher noise levels, performance on the noisy sigmoid network approaches that of the signum network, indicating that the noisy sigmoid is a valid (and differentiable) approximation for the signum. Right: As noise increases, the network adapts to sharpen its sigmoids, causing the first layer weights to increase, and the sigmoid output distributions to approach hard-limiters. Activation distributions were collected over the whole training set, with no noise added.

Here, the DVF results from on-off gas thrusters used for position and attitude control. Further details regarding the robot control application are presented in References [11,20–22].

4.1. Robot description

Experiments are performed using a mobile robot, shown in Figure 12, that operates in a horizontal plane, using air-cushion technology to simulate the drag-free and zero-g characteristics of space [20]. The three degrees-of-freedom (x, y, ψ) of the base are controlled using eight thrusters positioned around its perimeter in a horizontal plane near the centre of the robot. The thrusters are difficult to see in Figure 12, but their layout is shown in Figure 13. The on-off thrusters substantially complicate the control design, due to their discontinuous nature and the fact that each thruster simultaneously produces both a net force and torque. In addition to the NN-based methods mentioned here, a conventional control scheme based on a lookup table has been developed and implemented on the robot [20].

The overall objective is to use a set of eight full-on, full-off air jet thrusters to approximate a continuous-valued force vector that is commanded by the position/attitude control law of the mobile robot. The NN determines the combination of thrusters to fire that will generate a resultant force vector as close as possible to that commanded. This 'thruster-mapping' function and thruster layout are shown in Figure 13.

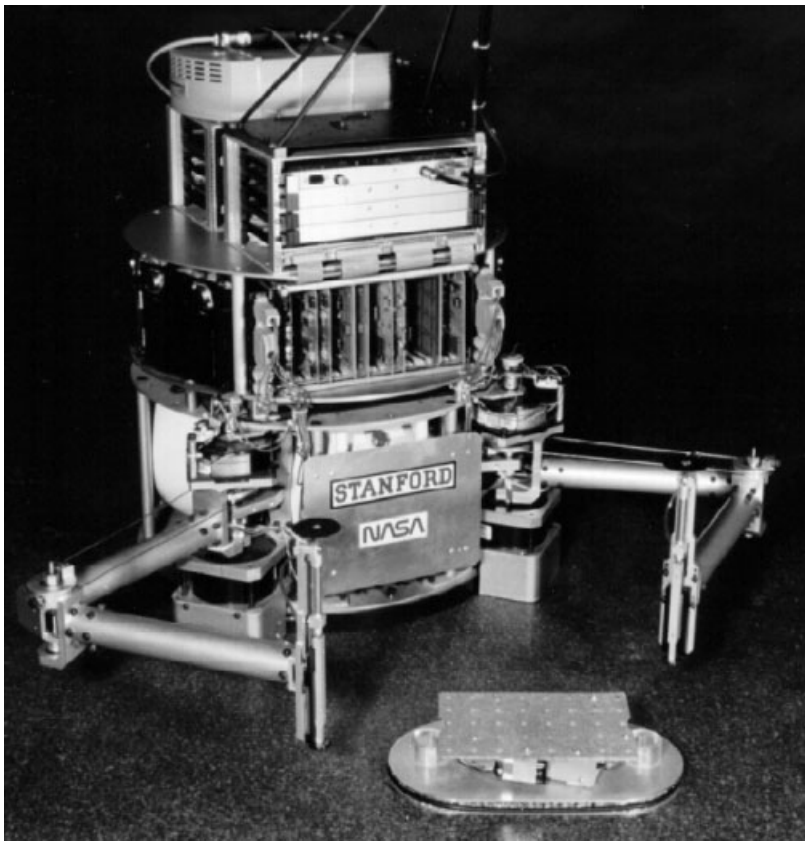

Figure 12. Stanford free-flying space robot. This highly autonomous mobile robot operates in the horizontal plane, using an air-cushion suspension to simulate the drag-free and zero-g characteristics of space. It is a fully self-contained laboratory-prototype of a free-flying space robot complete with on-board gas, thrusters, electrical power, multi-processor computer system, camera, wireless Ethernet data/communications link, and two cooperating manipulators.

4.2. Indirect training, application of noisy sigmoids

The training structure used to develop a NN solution to this thruster mapping problem is summarized in Figure 14. The neural network indicated is a standard sigmoid network containing no DVFs.

In this indirect training method, the NN must optimize the mapping through experimentation (in simulation) with the plant model. This requires backpropagation of error through the discontinuous thrusters, motivating development of the noise injection method presented in this paper.

Training without this noise-injection technique produces large errors, because the discrete-valued nature of the thrusters is not enforced during network training, and large round-off errors result at run-time. For example if one unit of thrust is requested in the $+x$ direction,

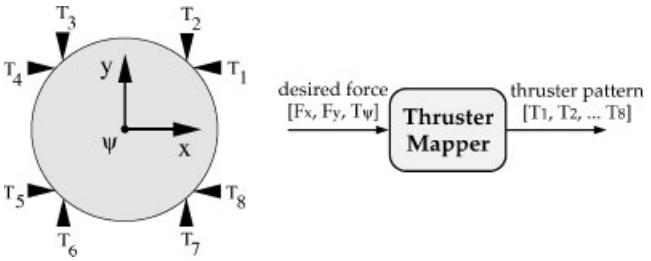

Figure 13. Thruster mapping, problem definition. At every sample period, the Thruster Mapper takes a desired force vector, $[F_{x\mathrm{des}}, F_{y\mathrm{des}}, \tau_{\psi\mathrm{des}}]$, and finds the thruster settings, $[T_1, T_2, \ldots, T_8]$, to minimize a specified cost function. The on-off thrusters and coupling between forces and torque make this problem difficult. This mapping is calculated several times per second, motivating the development of a nonlinear approximate solution (implemented as a neural network) that can run in real time.

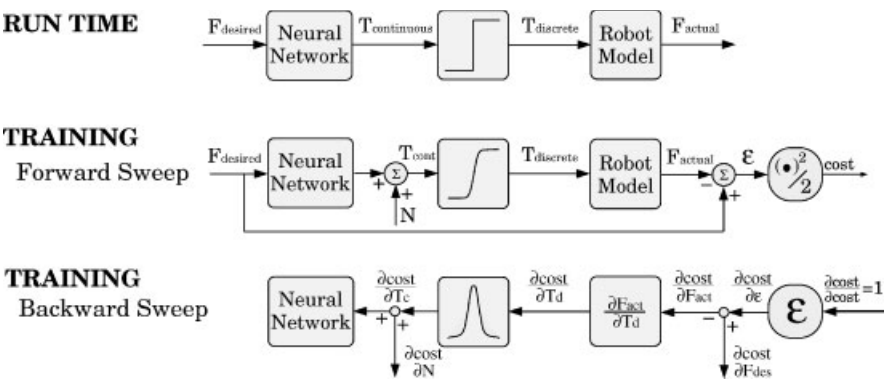

Figure 14. Thruster mapping, indirect training method.

during training, the network will set T_4 and T_5 to $+0.5$; but at run-time, for requested forces near 1.0, T_4 and T_5 are likely to both be 0 or both be 1, resulting in a large error. This situation is similar to the example presented in Section 2.2.

Figure 15 shows the result of indirect training with two differentiable thruster models. In both cases, the problem has been simplified by considering pairs of opposing bi-level thrusters as single tri-level thrusters. During training with the continuous thruster models, the NN produces a mapping with a very low error, which is not plotted here. However, when the continuous thrusters are replaced with discrete thrusters at run-time, the error is large, and is the 'thruster mapping error' plotted in the right half of Figures 15 and 16. The errors are high because the NN was optimized using outputs that would be unavailable at run-time, resulting in round-off error. The direct-training performance, also shown on the plots, represents the best result possible, limited by the functional complexity of the 3-10-4 layered network. In Figures 15 and 16, each dot represents the final performance after a 10 000 epoch training run. The shaded regions represent mean $\pm\,\sigma$ performance for ten runs.

Figure 16 shows the results when the thrusters are modelled by *noisy* tri-level sigmoids. With $N = 0$, error is high, corresponding to the data in Figure 15, but as noise increases, performance approaches the lower bound set by the functional complexity of the NN.

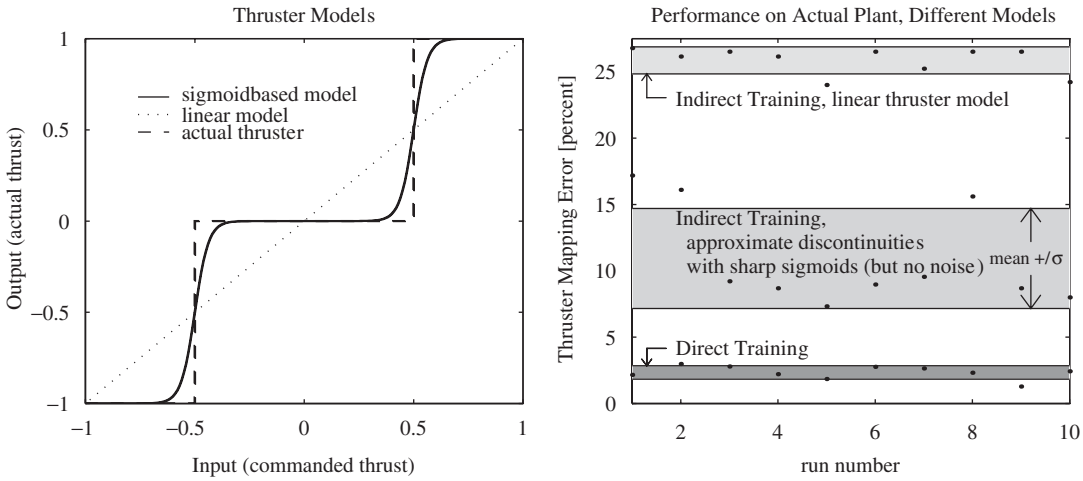

Figure 15. Results of indirect training, two differentiable thruster models. The sigmoid-based approximation (without noise) is better than the linear model, but has limited performance. The results from direct training represent a lower limit for comparison. Mapping error is average percent error above the optimal mapping (which results from an exhaustive search of all possible thruster combinations). The shaded areas represent the [mean $\pm$ σ] for 10 different runs. 3-10-4 layered networks were used.

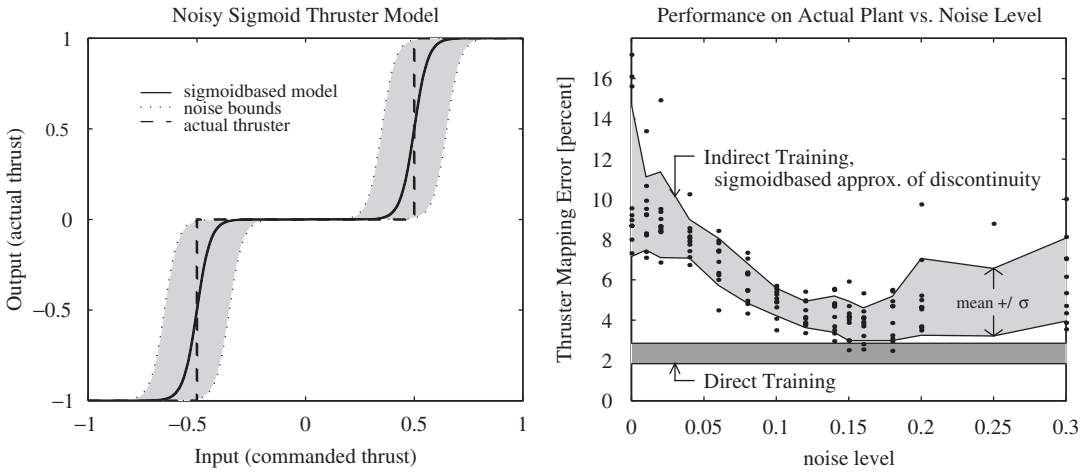

Figure 16. Results of indirect training, noisy tri-level sigmoid thruster model. Left: the sigmoid sharpness factor (slope at the midpoint = 4) and noise level (0.15) for the noisy tri-level sigmoid appear to be intuitively correct. Right: as noise increases, performance approaches that of the network trained directly (emulating the optimal mapping), with best performance at a noise level of about 0.15. 3-10-4 layered networks were used.

The best value for in this application seems to be around 0.15, and the resulting noisy sigmoid is shown in the left half of Figure 16. Examining this figure, the sigmoid sharpness and noise levels seem to be set correctly according to intuition. As N increases beyond 0.2, error increases

as expected (the 'off' region of the sigmoid becomes blurred). The method is fairly robust to the noise value selected, and the effect of noise level on performance makes intuitive sense.

A good solution results when noise is added, because it prevents the network from using a solution that uses non-saturated portions of the tri-level sigmoid. Such a solution would give a nearly random output and high error during training. The training algorithm must find a solution that works well despite the noise addition. This means the expected value of the output must be well into the saturated regions to work consistently well. The results approximate the optimal solution very well, and work when the tri-level sigmoids are replaced with tri-level signums.

5. CONCLUSIONS

Noisy Backpropagation is an effective algorithm for gradient-based parameter optimization of systems containing discrete-valued functions. This new algorithm has been presented and shown to extend gradient-based parameter optimization to work with systems containing discrete-valued functions, despite the discontinuity that exists between discrete values. The modification to backpropagation is very small, simply requiring approximation of the discrete-valued functions with continuously differentiable ones, and the injection of noise into the smooth approximating function on the forward sweep. The noise injection is critical to ensuring that the noisy sigmoid behaves like a signum during training. Insensitivity to the type and magnitude of noise injected has been demonstrated in several applications.

Multi-layered networks of hard-limiters require simpler processing hardware than do multi-layered sigmoid networks. Sigmoid networks are commonly used, however, due to their increased functionality as well as the lack of a reliable training algorithm for signum networks. Multi-layered signum networks have now been successfully trained using Noisy Backpropagation in two different applications, demonstrating its usefulness in this area.

Application to a complex thruster-control problem, with implementation on a laboratory model of a free-flying space robot, has demonstrated the method's realizability and usefulness for on-off control problems.

ACKNOWLEDGEMENTS

This research was partially funded by NASA Coop-Agreement NCC 2-333-S18, Department of the Air Force Grant F49620-92-J-0329, and Hughes Aircraft Company Howard Hughes Doctoral Fellowship. The authors wish to thank the many students, staff and faculty at the Stanford Aerospace Robotics Laboratory that contributed to this research, most notably Dr. Marc Ullman who led the design and construction of the robot used in this research, and professor Robert H. Cannon, Jr. who helped direct this research.

REFERENCES

1. Rumelhart DE, Hinton GE, Williams RJ. Learning internal representations by error propagation. *Parallel Distributed Processing.* The MIT Press: Cambridge, MA 02142, 1986; 318.
2. Werbos PJ. Beyond Regression: new tools for prediction and analysis in the behavioral sciences. *PhD Thesis*, Harvard University, Cambridge, MA 02142, 1974.

3. Hoff ME Jr. Learning phenomena in networks of adaptive switching circuits. *PhD Thesis*, Stanford University, Stanford, CA 94305, 1962. *Tech. Rep.* 1556-1, Stanford Electron. Labs.
4. Widrow B, Lehr MA. 30 years of adaptive neural networks: Perceptron, MADALINE, and backpropagation. *Proceedings of the IEEE* 1990; **78**(9):1415–1442.
5. Rosenblatt F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books: Washington, DC, 1962.
6. Winter R, Widrow B. Madaline Rule II: a training algorithm for neural networks. *IEEE International Conference on Neural Networks* vol. 1 1988; 401–408.
7. Bartlett PL, Downs T. Using random weights to train multilayer networks of hard-limiting units. *IEEE Transactions on Neural Networks* 1992; **3**(2):202–210.
8. Widrow B. A study of rough amplitude quantization by means of Nyquist sampling theory. *IRE Transactions of the Professional Group on Circuit Theory* 1956; **CT-3**(4):266–276.
9. An G. The effects of adding noise during backpropagation training on generalization performance. *Neural Computation* 1996; **8**(3):643–674.
10. Murray AF, Edwards PJ. Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on Neural Networks* 1994; **5**(5):792–802.
11. Wilson E. Experiments in neural network control of a free-flying space robot. *PhD Thesis*, Stanford University, Stanford, CA 94305, 1995.
12. Wilson E. Experiments in neural network control of a free-flying space robot. *Proceedings of the Fifth Workshop on Neural Networks*: *Academic/Industrial/NASA/Defense* 1993; 204–209; SPIE. *Proceedings of the SPIE*, vol. 2204.
13. Wilson E. Backpropagation learning for systems with discrete-valued functions. *Proceedings of the World Congress on Neural Networks*, vol. 3, 1994; 332–339.
14. Bryson AE Jr., Ho YC. *Applied Optimal Control.* Hemisphere Publishing Corporation: New York, NY, 1975.
15. MATLAB is a registered trademark of The MathWorks, Inc., 24 Prime Park Way, Natick, MA 01760, 508-647-7000.
16. Wilson E. Matlab source code for noisy backpropagation example. Available at http://sun-valley.stanford.edu, 1997.
17. Weigend AS, Huberman BA, Rumelhart DE. Predicting the future: a connectionist approach. *International Journal of Neural Systems* 1990; **1**(3):193–209.
18. Wilson E, Rock SM. Experiments in control of a free-flying space robot using fully-connected neural networks. *Proceedings of the World Congress on Neural Networks* vol. 3, 1993; 157–162.
19. Wertz JR, editor. *Spacecraft Attitude Determination and Control.* Kluwer Academic Publishers: Boston, MA, 1978.
20. Ullman MA. Experiments in autonomous navigation and control of multi-manipulator, free-flying space robots. *PhD Thesis*, Stanford University, Stanford, CA 94305, 1993.
21. Wilson E, Rock SM. Neural network control of a free-flying space robot. *Simulation* 1995; **65**(2):103–115.
22. Wilson E, Rock SM. Reconfigurable control of a free-flying space robot using neural networks. *Proceedings of the American Control Conference*, 1995.

